УДК 004.021; 51-76
DOI: 10.25559/SITITO.15.201901.81-91

Реализация метода ветвей и границ для задачи восстановления матрицы расстояний между последовательностями ДНК

М. Э. Абрамян[1], **Б. Ф. Мельников**[2], **М. А. Тренина**[3*]
[1] Южный федеральный университет, г. Ростов-на-Дону, Россия
[2] Российский государственный социальный университет, г. Москва, Россия
[3] Тольяттинский государственный университет, г. Тольятти, Россия
[*] trenina.m.a@yandex.ru

Аннотация

В биоинформатике имеются две важные задачи для исследования эволюционных процессов: вычисление расстояния между последовательностями ДНК и восстановление матрицы расстояний между такими последовательностями для различных геномов, когда известны не все элементы этой матрицы. В силу большой размерности цепочек ДНК для решения первой задачи используются эвристические алгоритмы. Их основной недостаток состоит в том, что при использовании различных эвристических алгоритмов для вычисления расстояния между одной и той же парой цепочек ДНК получаются различные значения.

Для проведения сравнительного анализа этих эвристических алгоритмов был введён показатель badness для всех треугольников, образующихся в матрице ДНК, и в идеале он должен быть равен нулю. Этот показатель использовался в дальнейшем и для решения второй задачи. На основе того, что показатель badness должен быть равен нулю, разработан алгоритм восстановления матрицы расстояний между цепочками ДНК. Этот алгоритм оптимизирован применением в нём метода ветвей и границ. С помощью описываемого нами варианта этого метода выбирается такая последовательность вычисления неизвестных элементов, при которой значение показателя badness матрицы будет наименьшим.

Часть статьи посвящена подробному описанию алгоритмов. Самым важным среди вспомогательных алгоритмов является алгоритм выбора разделяющего элемента, т.е. того из неизвестных элементов, который мы восстанавливаем в первую очередь. Основной алгоритм, т.е. собственно восстановление матрицы ДНК методом ветвей и границ, включает описание задачи, состоящей из подзадач; каждая из последних является набором из преобразованной матрицы, последовательности уже восстановленных элементов исходной матрицы и множества тех ещё не заполненных элементов матрицы, которые нельзя выбирать на следующем шаге алгоритма. В статье также приведено описание программной реализации описанных алгоритмов.

Ключевые слова: метод ветвей и границ, последовательности ДНК, матрица расстояний, частично заполненная матрица, восстановление.

Для цитирования: *Абрамян М. Э., Мельников Б. Ф., Тренина М. А.* Реализация метода ветвей и границ для задачи восстановления матрицы расстояний между последовательностями ДНК // Современные информационные технологии и ИТ-образование. 2019. Т. 15, № 1. С. 81-91. DOI: 10.25559/SITITO.15.201901.81-91

Implementation of the Branch and Bound Method for the Problem of Recovering a Distances Matrix Between DNA Strings

M. E. Abramyan[1], B. F. Melnikov[2], M. A. Trenina[3*]
[1] Southern Federal University, Rostov-on-Don, Russia
[2] Russian State Social University, Moscow, Russia
[3] Togliatti State University, Togliatti, Russia
[*] trenina.m.a@yandex.ru

Abstract

Bioinformatics has two important problems for the study of evolutionary processes: calculating the distance between DNA sequences and restoring a matrix of distances sequences of different genomes when not all initial elements are known. Due to the large dimension of DNA chains, heuristic algorithms are used to solve the first problem. The main lack of them is that when using different heuristic algorithms to calculate the distance between the same pair DNA chains, we produce different values.
To make a comparative analysis of these heuristic algorithms, a badness index was introduced for all the triangles formed in the DNA matrix, and ideally it should be zero. This indicator was used in the future and to solve the second problem. Based on the fact that the badness indicator should be equal to zero, the algorithm of restoring the matrix of distances between DNA chains is developed. This algorithm is optimized using the branch and bound method. This method selects a sequence of calculations of unknown elements, in which the value of the badness matrix will be the smallest.
A part of the paper is devoted to the detailed description of algorithms. The most important among the auxiliary algorithms is the algorithm for selecting the separating element, i.e. the unknown element, which we recover first. The main algorithm, i.e. the actual recovering the DNA matrix by the method of branch and bound, includes a description of the task, which consists of subtasks containing a set of the transformed matrix, a sequence of already restored elements of the original matrix and the set of those not yet filled elements of the matrix, which cannot be selected in the next step of the algorithm. The paper also includes the program implementation of the described algorithms.

Keywords: branch and bound method; DNA sequences; distance matrix; partially filled matrix; recovery.

For citation: Abramyan M.E., Melnikov B. F., Trenina M. A. Implementation of the Branch and Bound Method for the Problem of Recovering a Distances Matrix Between DNA Strings // *Sovremennye informacionnye tehnologii i IT-obrazovanie* = Modern Information Technologies and IT-Education. 2019; 15(1):81-91. DOI: 10.25559/SITITO.15.201901.81-91

Введение

В биоинформатике существуют две важные задачи для исследования эволюционных процессов:

- вычисление расстояния между последовательностями ДНК [1, 2, 3];
- восстановление соответствующей матрицы расстояний, когда известны не все её элементы [4, 5].

Стоит отметить, что даже в задачах, на первый взгляд не связанных с ДНК, как, например, некоторые задачи теории кодирования [6], есть выход на этот класс проблем; более подробно об этом написано в [7]. В силу большой длины цепочек ДНК для решения первой задачи используются эвристические алгоритмы [8, 9, 10]. Основной их недостаток состоит в том, что при использовании различных алгоритмов для вычисления расстояния между одной и той же парой цепочек ДНК получаются различные значения.

Современные исследования в биоинформатике в основном направлены на использование известных эвристических алгоритмов вычисления расстояния между последовательностями ДНК в сравнительной геномике [11, 12], но при этом не проводится анализ этих алгоритмов.

Поэтому возникает задача оценки качества используемых метрик (расстояний) – и по результатам, полученным при решении этой задачи, можно делать выводы о применимости конкретного алгоритма подсчёта расстояний к различным прикладным исследованиям.

Для проведения сравнительного анализа этих эвристических алгоритмов был введён показатель badness для всех треугольников, образующихся в матрице ДНК, и в идеале он должен быть равен нулю [13, 14, 15, 16]. Этот показатель использовался в дальнейшем и для решения второй задачи. На основе того, что показатель badness желательно сделать как можно меньшим, разработан алгоритм восстановления матрицы расстояний между цепочками ДНК [17, 18].

Для оптимизации разработанного алгоритма в нём дополнительно применяется метод ветвей и границ. С помощью этого метода определение неизвестных элементов матрицы происходит в той последовательности, при которой значение показателя badness матрицы будет минимальным [19, 20].

Настоящая статья посвящена подробному рассмотрению реализации метода ветвей и границ при решении задачи восстановления матрицы расстояний между цепочками ДНК. Приведено описание основных классов и структур данных, используемых для реализации алгоритма.

При тестировании алгоритма в эталонной матрице восстановления соответствующей матрицы расстояний, заполненной с помощью алгоритма Нидлмана – Вунша, случайным образом обнулялось некоторое число элементов (мы для вычислительных экспериментов выбирали значения от 8 до 40 с шагом 8), а затем эти элементы восстанавливались с помощью разработанной программы. Анализ результатов вычислительного эксперимента проводился двумя способами:

- полученные значения сравнивались с соответствующими значениями эталонной матрицы;
- анализировался показатель badness полученной матрицы, значение которого должно быть минимальным.

В первом разделе кратко описана задача восстановления матрицы расстояний между последовательностями ДНК и основные подходы к её решению, подробное изложение которых было в работах [17, 18]. Приведён основной алгоритм метода ветвей и границ, а также вспомогательные алгоритмы для его реализации. Второй раздел настоящей статьи посвящен описанию подходов к программной реализации этих алгоритмов. В третьем разделе приведены результаты тестирования разработанных алгоритмов.

Описание алгоритмов

Формулировка задачи: дана (симметричная) матрица расстояний $N \times N$ между N последовательностями ДНК различных геномов, полученная каким-либо стандартным алгоритмом (например, Нидлмана – Вунша). Однако в этой матрице нам при её задании неизвестно некоторое число элементов. Такие матрицы мы называем неполностью определёнными [17].

Цель алгоритма: восстановление отсутствующих (изначально неизвестных) элементов. В процессе работы алгоритма строится семейство последовательностей восстанавливаемых элементов. Требуется выбрать из всех (или, по крайней мере, всех проанализированных) последовательностей оптимальную по некоторым показателям.

Определение неизвестного элемента a_{ij} происходит на основании предположения о том, что все треугольники, построенные на элементах a_{ij} a_{ki} a_{kj} должны быть «наиболее близкими» к равнобедренным остроугольным треугольникам; при этом в случае, когда известны две стороны треугольника; третья (неизвестная) сторона полагается равной большей из известных [17, раздел 4]. Так как подходящих значений k для одного неизвестного элемента a_{ij} может быть несколько, в качестве приближения для элемента a_{ij} ыбирается среднее арифметическое из значений, найденных для всех существующих треугольников.

Также для каждого треугольника в матрице используется показатель badness отхода этого треугольника от равнобедренного остроугольного

$$\sigma = \frac{a-b}{a}, \tag{1}$$

где a,b,c стороны треугольника, причём $a \geq b \geq c$ Для вычисления этого показателя могут быть использованы и другие подходы, эти формулы приведены в работах [14, 15, 16].

От показателя badness для треугольника мы переходим к аналогичному показателю для элемента матрицы: для каждого элемента, лежащего выше главной диагонали, рассматриваем все допустимые треугольники, построенные на нём, вычисляем показатель badness для каждого треугольника, после чего ставим в соответствие элементу $a_{ij}(i<j)$ ледующее значение:

$$\sigma_{ij} = \max_k \sigma_k, \tag{2}$$

где σ_k показатель badness допустимого треугольника, построенного на элементах a_{ij} a_{ki} a_{kj} 19, раздел 3].

Для всей матрицы ДНК показатель total_badness определяется как сумма значений badness по всем элементам (ненулевым) элементам, лежащим выше главной диагонали:

$$\delta = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} \sigma_{ij}. \tag{3}$$

Здесь и в дальнейшем при описании алгоритмов предполагается, что строки и столбцы матрицы индексируются от 0.

Матрицу расстояний между последовательностями ДНК необ-

ходимо восстановить так, чтобы показатель total_badness, определенный в (3), принимал наименьшее значение.

В рассматриваемой нами задаче в качестве пространства допустимых решений берутся *все возможные последовательности определения неизвестных элементов* верхней половины матрицы. Построение последовательностей выполняется методом ветвей и границ [21, 22], причём в качестве разделяющих элементов выбираются незаполненные (нулевые) элементы матрицы.

Выбор разделяющего элемента происходит на основе следующей предложенной нами вспомогательной эвристики. Сначала необходимо:

- найти количество kol_{ij} аких k для которых a_{ki} a_{kj} звестны и $k \neq i$. $k \neq j$;
- вычислить разность $|a_{ki} - a_{kj}|$ (4)

Мы отбираем элементы, для которых kol_{ij} инимает наибольшее значение, а потом среди них в качестве разделяющего элемента выбираем тот, для которого разность (4) является наибольшей [18, раздел 3].

Ветвление на каждом разделяющем элементе происходит следующим образом: для выбранного разделяющего элемента все последовательности разделяются на две группы:

- содержащие в данной позиции этот элемент;
- не содержащие в данной позиции этого элемента.

Граница для каждой формируемой последовательности определяемых элементов полагается равной значению total_badness для матрицы, включающей уже определённые элементы последовательности. Таким образом, заполнение матрицы новыми элементами в процессе формирования последовательности приводит к увеличению (точнее, неуменьшению) границы для данной последовательности [19, раздел 3].

Приведём формализованное описание алгоритма восстановления матрицы ДНК методом ветвей и границ, включающее описание ряда вспомогательных алгоритмов.

Алгоритм 1. Выбор разделяющего элемента (т. е. «наилучшего» из неизвестных элементов, который восстанавливается в первую очередь).

Вход: Неполностью определённая матрица $A = \| a_{ij} \|$

Описание алгоритма.

1) Для каждого *нулевого* элемента $a_{ij}, i = 1,\ldots n-1, j = 0,\ldots,i-1$ определяются две характеристики: kol_{ij} количество треугольников, построенных на данном элементе) и mas_{ij} максимальная разность между известными элементами, входящих в треугольник, наряду с элементом a_{ij}

kol_{ij} =0

mas_{ij} =0

for k = 0 to $n-1$ do

if ($k <> i$)and ($k <> j$)and (a_{ki} <>0 and (a_{kj} <>0) then

begin

kol_{ij} += 1;

mas_{ij} =Max(mas_{ij} , Abs(a_{ki} - a_{kj}));

end;

2) В качестве разделяющего неизвестного элемента выбирается тот, для которого значение kol_{ij} вляется максимальным (и при этом не равным 0), а в случае нескольких таких элементов – тот, для которого mas_{ij} является наибольшим.

3) Если все значения kol_{ij} авны 0, то наилучший элемент выбрать нельзя.

Выход: найденные значения kol_{ij} mas_{ij}

Алгоритм 2. Определение значения выбранного разделяющего элемента a_{ij} *Вход:* 1. Неполностью определённая матрица $A = a_{ij}$ 2. Значение kol_{ij} вычисленное в алгоритме 1.

Описание алгоритма.

for k = 0 to $n-1$ o

if ($k <> i$)and ($k <> j$)and (a_{ki} <>0) and (a_{kj} <>0) then

$a_{ij} += Max(a_{ki}, a_{kj})$;

$a_{ij} = a_{ij} / kol_{ij}$

Выход: значение для разделяющего элемента a_{ij}

Алгоритм 3. Вычисление показателя $badness_{ij}$ ля *ненулевого* элемента a_{ij} $(i < > j)$

Вход: 1. Неполностью определённая матрица $A = \| a_{ij} \|$

Описание алгоритма.

$badness_{ij}$ =0

for $k := 0$ to n-1 do

if ($k <> i$)and ($k <> j$)and (a_{ki} <>0) and (a_{kj} <>0) then

begin

a_{ij}, a_{ki}, a_{kj} сортируются по убыванию

и обозначаются соответственно $x, y, z (x > y > z)$

$badness_{ij} = Max(badness_{ij}, (x-y)/x)$

end;

Выход: значение $badness_{ij}$ ля ненулевого элемента a_{ij}

Алгоритм 4. Вычисление показателя Total_badness для матрицы в целом.

Вход: 1. Неполностью определённая матрица $A = \| a_{ij} \|$

Описание алгоритма.

Суммируются показатели $badness_{ij}$ ля всех известных элементов, лежащих выше главной диагонали (т. е. с индексами $i = 0,\ldots n-2, j = i+1,\ldots,n-1$

Выход: значение Total_badness для матрицы A

Основной алгоритм. Восстановление матрицы ДНК методом ветвей и границ.

Под *подзадачей* понимается набор из преобразованной матрицы, последовательности Yes уже восстановленных элементов исходной матрицы и множества No тех еще не заполненных элементов матрицы, которые *нельзя выбирать* на следующем шаге алгоритма.

Вход: 1. Неполностью определенная матрица $A = \| a_{ij} \|$ (все

равные нулю элементы вне главной диагонали считаем неизвестными).

Описание алгоритма.

1. Инициализация: список подзадач полагается пустым, псевдооптимальное решение opt полагается отсутствующим (opt = null).
2. Добавление в список подзадач подзадачи, соответствующей исходной матрице с пустыми наборами Yes и No.
3. Если список подзадач пуст или время работы превысило допустимое, то вывод opt и завершение работы основного алгоритма.
4. Выбор новой подзадачи из списка подзадач (выбирается подзадача с минимальным числом неопределенных (нулевых) элементов, а среди таких подзадач – подзадача с минимальным значением матрицы Total_badness).
5. Выбор наилучшего неизвестного элемента (из числа не входящих в набор No) в выбранной подзадаче (алгоритм 1).
6. Если наилучший элемент нельзя выбрать, то переход к шагу 4.
7. Создание новой подзадачи, совпадающей с текущей, добавление к её множеству No выбранного элемента и помещение новой подзадачи в список подзадач.
8. Преобразование текущей подзадачи: определение значения для выбранного элемента (алгоритм 2), пересчет показателей $badness_{ij}$ алгоритм 3) и Total_badness (алгоритм 4) для нового варианта матрицы, очистка множества No, добавление к последовательности Yes выбранного элемента.
9. Если матрица текущей подзадачи не содержит неизвестных элементов, то сравнение её характеристики Total_badness с аналогичной характеристикой псевдооптимального решения opt (при его наличии).
10. Если характеристика Total_badness текущей подзадачи меньше (или решение opt ещё не определено), то opt полагается равным текущей подзадаче; в противном случае решение opt не изменяется; в любом случае после этого выполняется переход к шагу 3.
11. Если характеристика Total_badness матрицы текущей подзадачи больше соответствующей характеристики решения opt, то переход к шагу 3 (т.е. выполняется досрочное отсечение неоптимальных подзадач).
12. Переход к шагу 5.

Выход: превдооптимальное решение *opt*.

Реализация алгоритмов

Алгоритм реализован на языке C# 6.0 (среда разработки Visual Studio 2017).

Для реализации алгоритма восстановления матрицы ДНК методом ветвей и границ использовались 3 основных класса:

`Matrix` – восстанавливаемая *матрица* вместе с дополнительными характеристиками;

`Subtask` – *подзадача*, являющаяся потомком **`Matrix`** и содержащая дополнительные свойства Yes и No;

`Task` – класс, реализующий *основной алгоритм* в полном объёме и включающий поля **`subtasks`** (набор подзадач) и **`opt`** (текущее псеводооптимальное решение).

Опишем каждый из этих классов.

Для хранения элементов матрицы в классе Matrix используется невыровненный массив matr типа ItemVB[][], содержащий элементы матрицы, расположенные выше главной диагонали, причём объект типа ItemVB включает как значение Value элемента матрицы, так и его характеристику Badness. Для упрощения доступа к любому элементу матрицы в классе Matrix реализовано индексированное свойство this[i,j]:

```
public double this[int i, int j] noSelect
{
      get
      {
        return i > j ? matr[i - 1][j].Value :
               i < j ? matr[j - 1][i].Value : 0.0;
      }
}
```

Для ускорения обработки неизвестных элементов матрицы (алгоритм 1) в класс Matrix включена коллекция z типа SortedSet<ItemZ>, содержащая данные об этих элементах, а именно, объекты ItemZ с полями:

- *X, Y* – индексы *i* и *j* нулевого элемента a_{ij}
- *K* – характеристика kol_{ij}
- *M* – характеристика mas_{ij}

Для объектов ItemZ определена *операция сравнения*, упрощающая выбор «наилучшего» неизвестного элемента. Сравнение реализовано в соответствии с шагом 2 алгоритма 1: элемент *a* больше элемента *b*, если *a.K* > *b.K* или, в случае равенства полей *K*, если *a.M* > *b.M* (если равны поля *K* и *M*, то элементы упорядочиваются в соответствии с их индексами в матрице).

Коллекция **`z`** типа **`SortedSet<ItemZ>`** хранит данные о неизвестных элементах в *отсортированном* порядке; при этом для получения «наилучшего» элемента достаточно обратиться к элементу **`z.Max`**. В методе **`SelectNextZ`**, предназначенном для получения очередного наилучшего элемента, предусмотрен параметр, позволяющий исключить из рассмотрения те неизвестные элементы, которые нельзя выбирать для данной подзадачи (см. шаг 5 основного алгоритма):

```
public ItemZ? SelectNextZ(SortedSet<ItemZ>
noSelect)
 {
 ItemZ? res = null;
 z.ExceptWith(noSelect);
 if (z.Count > 0)
 res = z.Max;
 z.UnionWith(noSelect);
 return res;
}
```

Важным методом класса **`Matrix`** является метод **`ZtoNZ(ItemZ item)`**, определяющий значение очередного восстанавливаемого элемента item (алгоритм 2) и пересчитывающий после этого все характеристики элементов матрицы (алгоритмы 3 и 4). При этом алгоритм 3 нахождения характеристики $badness_{ij}$ ealизуется в полном объёме (с использованием цикла) только для восстанавливаемого элемента, тогда как для остальных известных элементов выполняется лишь

корректировка ранее вычисленной характеристики Badness с учётом того, что в матрице появляется новый элемент (подобная корректировка не требует применения цикла). В этом же методе выполняется корректировка характеристик kol_{ij} mas_{ij} ставшихся неизвестных элементов (хранящихся в коллекции z) с учётом нового известного элемента; эта корректировка, в отличие от полного варианта алгоритма 1, также не требует применения цикла.

```
public Point ZtoNZ(ItemZ item)
{
  int x = item.X;
  int y = item.Y;
  double v = 0.0;              //алгоритм 2
  for (int k = 0; k < N; ++k)
    if (k != x && k != y && this[k, x] != 0 &&
this[k, y] != 0)
      v += Math.Max(this[k, x], this[k, y]);
  v /= item.K;
  if (v == 0)
    v = double.Epsilon;
  double bad = badness(x, y, v);
  z.Remove(item);
  if (x > y) matr[x - 1][y].Value = v;
  else if (x < y) matr[y - 1][x].Value = v;
    SetBadness(x, y, bad);
          // пересчет badness с учётом добав-
ленного элемента (алгоритм 3)
  for (int i = 0; i < N-2; ++i)
    for (int j = i+1; j < N-1; ++j)
    {
      double v0 = this[i, j];
     if (v0 == 0.0 || x == i && y == j) continue;
      double bad0 = 0.0;
      if (x == i)
     { if (this[j, y] != 0.0) bad0 = badness3(v0,
v, this[j, y]); }
      else if (x == j)
     { if (this[i, y] != 0.0) bad0 = badness3(v0,
v, this[i, y]); }
      else if (y == i)
     { if (this[j, x] != 0.0) bad0 = badness3(v0,
v, this[j, x]); }
      else if (y == j)
     { if (this[i, x] != 0.0) bad0 = badness3(v0,
v, this[i, x]); }
      if (bad0 > Badness(i, j))    //алгоритм 4
      {
        total_badness += bad0 - Badness(i, j);
        SetBadness(i, j, bad0);
      }
    }
  var res = new Point(x, y);
  total_badness += bad;
       // пересчёт характеристик оставшихся ну-
левых элементов
       // с учётом добавленного элемента (алго-
ритм 1)
  foreach (var e in z.ToArray())
  {
    int i = e.X;
    int j = e.Y;
    int kol0 = e.K;
    double mas0 = e.M;
    double m = 0.0;
    if (x == i)
    { if (this[j, y] != 0.0) {++kol0; m = Math.
Abs(v - this[j, y]);} }
    else if (x == j)
    { if (this[i, y] != 0.0) {++kol0; m = Math.
Abs(v - this[i, y]);} }
    else if (y == i)
    { if (this[j, x] != 0.0) {++kol0; m = Math.
Abs(v - this[j, x]);} }
    else if (y == j)
    { if (this[i, x] != 0.0) {++kol0; m = Math.
Abs(v - this[i, x]);} }
    if (m > mas0) mas0 = m;
    if (kol0 != e.K || mas0 != e.M)
    {
      z.Remove(e);
      z.Add(new ItemZ(i, j, kol0, mas0));
    }
  }
  return res;
}
```

Признаком того, что в матрице восстановлены все неизвестные элементы, является отсутствие элементов в коллекции z:

```
    public bool IsReady
      { get { return z.Count == 0; } }
```

Класс **Subtask** является потомком класса **Matrix**; кроме самой матрицы и её характеристик он содержит порядковый номер подзадачи (поле Id), сведения о «восстановленных» элементах (поле **Yes**, являющееся коллекцией типа **List<Point>**) и сведения о тех неизвестных элементах, которые *нельзя восстанавливать* на текущем шаге (поле No, являющееся коллекцией типа **SortedSet<ItemZ>**). Заметим, что наличие поля No, в соответствие с МВГ, позволяет избежать повторной генерации одинаковых последовательностей восстановленных элементов. Структура **Point**, используемая для хранения данных о восстановленном элементе, содержит индексы элементов матрицы — поля X и Y.

Для объектов класса **Subtask** определена *операция сравнения*, позволяющая выбирать очередную подзадачу из списка подзадач в соответствии с условиями шага 4 основного алгоритма (см. далее описание класса **Task**): подзадача *a* меньше подзадачи *b*, если число нулевых элементов в матрице *a* меньше числа нулевых элементов в матрице *b*, а в случае их равенства — если характеристика Total_badness матрицы *a* меньше характеристики Total_badness матрицы *b* (при равенстве обеих указанных характеристик сравниваются порядковые номера подзадач).

Основным методом класса **Subtask** является метод **MakeRight()**, охватывающий шаги 5–8 основного алгоритма:

```
public Subtask MakeRight()
{
```

```
  ItemZ? next = SelectNextZ(No);
  SortedSet<ItemZ> NoRight = No;
  No = new SortedSet<ItemZ>();
  if (next == null)
    return null;
  Subtask right = new Subtask(this);
  NoRight.Add((ItemZ)next);
  right.No = NoRight;
  right.Yes.AddRange(Yes);
  var item = ZtoNZ((ItemZ)next);
  Yes.Add(item);
  return right;
}
```

В данном методе для исходной подзадачи выбирается очередной восстанавливаемый элемент (функция **SelectNextZ**), создается новая подзадача с той же матрицей, и набором No, включающим выбранный элемент (см. оператор **NoRight.Add((ItemZ)next)**), в исходной подзадаче определяются характеристики для выбранного элемента и пересчитываются характеристики других элементов (функция **ZtoNZ((ItemZ)next)**), а выбранный элемент добавляется в набор **Yes** (оператор **Yes.Add(item)**). Метод возвращает созданную подзадачу, если в исходной подзадаче удалось выбрать очередной восстанавливаемый элемент, или **null**, если выбрать элемент не удалось.

Класс **Task** обеспечивает перебор подзадач. Он содержит набор подзадач (поле **subtasks** типа **SortedSet<Subtask>**) и текущее псевдооптимальное решение (поле **opt** типа **Subtask**).

При создании данного класса указывается исходная матрица и параметр **maxStepCount**, определяющий максимальное число анализируемых последовательностей.

Основным методом класса **Task** является метод **BigStep(int id)**, обеспечивающий генерацию очередной последовательности восстанавливаемых элементов, причём в ходе генерации набор подзадач пополняется новыми подзадачами, ожидающими своей обработки.

```
public bool BigStep(int id)
        // каждый "большой шаг" завершается построением
       // очередной последовательности или получением
        // последовательности со слишком большой нижней границей
{
  Subtask current = subtasks.Min;
  subtasks.Remove(current);
  current.Id = id;
  while (true)
  {
    if (current.IsReady)
       // построение закончено
    {
      if (opt == null || current.Total_badness < opt.Total_badness)
      // найдено новое псевдооптимальное решение
      {
        opt = current;
                OnEndStep?.Invoke(StepResult.Opt, current);
      }
      else
            OnEndStep?.Invoke(StepResult.NonOpt, current);
      return true;
    }
    if (opt != null && current.Total_badness > opt.Total_badness)
       // получена последовательность с большой нижней границей
    {
            OnEndStep?.Invoke(StepResult.NonOpt, current);
       return true;
    }
    Subtask right = current.MakeRight();
    if (right != null)
      subtasks.Add(right);
    else
      return false;
  }
}
```

В этом методе:

- из набора подзадач извлекается очередная подзадача **current** (используется метод **subtasks.Min**, позволяющий выбрать очередную подзадачу в соответствии с условиями, указанными в шаге 4 основного алгоритма), идентификатор **Id** этой задачи изменяется на значение параметра **id** метода **BigStep**;
- для выбранной подзадачи в цикле вызывается метод **MakeRight** (см. выше его описание), причем возвращаемая этим методом новая подзадача добавляется в набор подзадач;
- обработка подзадачи **current** (а вместе с ней и работа метода **BigStep**) завершается либо после определения в ней всех нулевых элементов, либо в случае, если ее характеристика Total_badness превысит соответствующую характеристику текущего псевдооптимального решения **opt** (см. шаги 9 и 10 основного алгоритма). При этом метод **BigStep** возвращает значение true.

Если при очередном вызове метода **MakeRight** будет возвращено значение **null** (означающее, что не удалось выбрать очередной восстанавливаемый элемент матрицы), то метод **BigStep** немедленно завершается со значением **false**.

В результате работы метода **BigStep** может быть изменено псевдооптимальное решение **opt**. С ситуацией завершения обработки очередной подзадачи может быть связан *обработчик события* **OnEndStep**, в котором можно определить дополнительные действия (например, вывести информацию о полученном варианте решения или откорректировать статистические данные). Обработчик события **OnEndStep** принимает два параметра: **stepResult** перечислимого типа **StepResult** со значениями **Opt** и **NonOpt** (данный параметр позволяет определить, является ли построенное решение псевдооптимальным) и **current** типа **Subtask**, содержащий построенное решение.

Метод **BigStep** вызывается в методе **Run()** класса **Task**:

```
      public int Run()
      {
        int i = 0;
            while (subtasks.Count > 0 && i <
maxStepCount)
          if (BigStep(i)) i++;
        return i;
}
```

Данный метод выполняется, пока число обработанных подзадач не достигнет значения **maxStepCount** или пока не исчерпаются все подзадачи. Он возвращает число обработанных подзадач.

Для запуска алгоритма МВГ достаточно вызвать конструктор класса **Task**, передав ему исходную матрицу (либо в виде объекта типа **Matrix**, либо в виде двумерного массива **double[,]**) и количество шагов **maxStepCount**, настроить, при необходимости, для созданного объекта **Task** обработчик события **OnEndStep** и вызвать для этого объекта метод **Run**.

Тестирование алгоритмов

Для проведения вычислительного эксперимента мы взяли матрицу, полученную в результате применением алгоритма Нидлмана-Вунша [23, 24]. Этот алгоритм был применён к цепочкам мДНК различных животных, взятых из банка данных NCBI[1]; при этом были взяты секвенированные цепочки мДНК для одного представителя каждого из 28 отрядов млекопитающих (классификацию млекопитающих выбираем согласно [25]. В ней случайным образом обнулялось некоторое число элементов (мы для вычислительных экспериментов выбирали значения от 8 до 40 с шагом 8), а затем эти элементы восстанавливались с помощью разработанной программы и сравнивались с эталонными значениями.

Для каждого количества нулей генерировалось 10 матриц. Для каждой матрицы алгоритм запускался при трех значениях максимального количества обработанных последовательностей: 1000, 10000 и 99000.

При каждом запуске определялась статистика следующего вида:

- *Zeros* – количество неизвестных элементов;
- *Sec* – время работы алгоритма (в секундах);
- *All* – количество обработанных последовательностей;
- *Opt* – количество найденных псевдооптимальных последовательностей (т.е. сколько раз происходило обновление псевдооптимального решения);
- *Cut* – количество отсечений, выполненных до завершения построения очередной последовательности;
- *TaskID* – порядковый номер построенной последовательности;
- *Badness* – соответствующая характеристика матрицы для данной последовательности;
- *Max* – максимальное отклонение построенных элементов от соответствующих элементов «эталонной» матрицы;
- *Avr* – среднеквадратическое отклонение этих элементов (невязка).

Набор из четырех последних характеристик (от TaskId до Avr) рассматривался для трех вариантов последовательностей:

- *1st* – самой первой построенной последовательности (она же «первая оптимальная»);
- *Min* – итоговой оптимальной последовательности (т.е. «самой хорошей из найденных»);
- *Max* – «самой плохой» из полученных последовательностей (с наибольшим значением *badness*).

Далее приведен пример обработки одной и той же матрицы с меньшим и большим числом проанализированных последовательностей (соответственно 1000 и 99000):

Таблица 1. Результаты тестирования программы восстановления матрицы расстояний между цепочками ДНК методом ветвей и границ
Table 1. The results of testing the program for restoring the matrix of distances between DNA chains by the branch and bound method

Zeros	*Sec*	*All*	*Opt*	*Cut*		*TaskId*	*Badness*	*Max*	*Avr*
40	0,1	1000	17	83	*1st:*	0	67,681	0,105	0,00853
					Min:	586	67,429	0,105	0,00847
					Max:	247	67,755	0,105	0,00856
40	14,4	99000	19	18866	*1st:*	0	67,681	0,105	0,00853
					Min:	40906	67,429	0,105	0,00847
					Max:	36847	67,793	0,105	0,00856

Стоит отметить, что значения как «плохих», так и «хороших» вариантов очень близки между собой и, кроме того, близки к первому из найденных вариантов.

Заключение

Разработанный и реализованный метод ветвей и границ для оптимизации разработанного ранее метода восстановления матрицы расстояний между последовательностями ДНК позволяет проводить вычисление неизвестных элементов матрицы в такой последовательности, при которой характеристика матрицы Total_badness принимает наименьшее значение. Минимизация этой характеристики приводит к тому, что среднеквадратическое отклонение найденных элементов от эталонных значительно уменьшается. При этом время работы алгоритма увеличивается незначительно.

Список использованных источников

[1] *Toppi J.* et al. A new statistical approach for the extraction of adjacency matrix from effective connectivity networks / J. Toppi, F.De Vico Fallani, M. Petti, G. Vecchiato, A.G. Maglione, F. Cincotti, S. Salinari, D. Mattia, F. Babiloni, L. Astolfi // Proceedings of the 35th Annual International Conference of

[1] Nucleotide database [Электронный ресурс] // NCBI. URL: http://www.ncbi.nlm.nih.gov/nuccore (дата обращения: 15.12.2018).

the IEEE Engineering in Medicine and Biology Society (EMBC), Osaka, 2013. Pp. 2932-2935. DOI: 10.1109/EMBC.2013.6610154

[2] *Zhou J., Zhong P., Zhang T.* A Novel Method for Alignment-free DNA Sequence Similarity Analysis Based on the Characterization of Complex Networks // Evolutionary Bioinformatics. 2016. № 12. Pp. 229-235. DOI: 10.4137/EBO.S40474

[3] *Alexeev N., Aidagulov R., Alekseyev M.A.* A computational method for the rate estimation of evolutionary transpositions // Bioinformatics and Biomedical Engineering. IWBBIO 2015. Lecture Notes in Computer Science / F. Ortuño, I. Rojas (eds). Springer, Cham, 2015. Vol. 9043. Pp. 471-480. DOI: 10.1007/978-3-319-16483-0_46

[4] *Евдокимова Н.И., Кузнецов А.В.* Локальные шаблоны в задаче обнаружения дубликатов // Компьютерная оптика. 2017. Т. 41, № 1. С. 79-87. DOI: 10.18287/2412-6179-2017-41-1-79-87

[5] *Eckes B., Nischt R., Krieg T.* Cell-matrix interactions in dermal repair and scarring // Fibrogenesis and Tissue Repair. 2010. Vol. 3, Issue 4. 11 p. DOI: 10.1186/1755-1536-3-4

[6] *Korabelshchikova S.Y., Melnikov B.F., Pivneva S.V., Zyablitseva L.V.* Linear codes and some their applications // Journal of Physics: Conference Series. 2018. Vol. 1096. Pp. 012174. DOI: 10.1088/1742-6596/1096/1/012174

[7] *Korabelshchikova S.Y., Melnikov B.F., Pivneva S.V., Zyablitseva L.V.* Linear error correcting codes and their application in DNA analysis // Proceedings of the 4th International Conference on Information Technology and Nanotechnology (ITNT-2018). Samara, 2018. Pp. 2095-2100. URL: https://elibrary.ru/item.asp?id=34894963& (дата обращения: 15.12.2018).

[8] *Melnikov B., Radionov A., Gumayunov V.* Some Special Heuristics for Discrete Optimization Problems // Proceedings of the Eighth International Conference on Enterprise Information Systems: Databases and Information Systems Integration (ICEIS- 2006), Paphos, Cyprus, May 23-27, 2006. Pp. 360-364.

[9] *Melnikov B.* Discrete optimization problems - some new heuristic approaches // Proceedings of the Eighth International Conference on High-Performance Computing in Asia-Pacific Region (HPCASIA'05), Beijing, 2005. Pp. 8 pp.-82. DOI: 10.1109/HPCASIA.2005.34

[10] *Melnikov B.F., Melnikova E.A., Pivneva S.V., Dudnikov V.A., Davydova E.V.* Geometric and game approaches for some discrete optimization problems // CEUR Workshop Proceedings. 2018. Vol. 2212. Pp. 312-321. URL: http://ceur-ws.org/Vol-2212/paper42.pdf (дата обращения: 15.12.2018).

[11] A promoter-level mammalian expression atlas / A. R. R. Forrest [et al.] // Nature. 2014. Vol. 507. Pp. 462-470. DOI: 10.1038/nature13182

[12] *Klyosov A.A., Faleeva T.* Excavated DNA from Two Khazar Burials // Advances in Anthropology. 2017. Vol. 7. Pp. 17-21. DOI: 10.4236/aa.2017.71002

[13] *Мельников Б.Ф., Пивнева С.В., Трифонов М.А.* Мультиэвристический подход к сравнению качества определяемых метрик на множестве последовательностей ДНК // Современные информационные технологии и ИТ-образование. 2017. Т. 13, № 2. С. 89-96. DOI: 10.25559/SITITO.2017.2.235

[14] *Melnikov B.F., Pivneva S.V., Trifonov M.A.* Comparative analysis of the algorithms of calculating distances of DNA sequences and some related problems // Proceedings of the Third International Conference on Information Technology and Nanotechnology (ITNT-2017). Samara, 2017. Pp. 1640-1645. URL: https://elibrary.ru/item.asp?id=29267020 (дата обращения: 15.12.2018).

[15] *Melnikov B.F., Pivneva S.V., Trifonov M.A.* Various algorithms, calculating distances of DNA sequences, and some computational recommendations for use such algorithms // CEUR Workshop Proceedings. 2017. Vol. 1902. Pp. 43-47. URL: http://ceur-ws.org/Vol-1902/paper8.pdf (дата обращения: 15.12.2018).

[16] *Melnikov B.F., Melnikova E.A., Pivneva S.V., Trenina M.A.* An approach to analysis of the similarity of DNA-sequences // CEUR Workshop Proceedings. 2018. Vol. 2212. Pp. 63-72. URL: http://ceur-ws.org/Vol-2212/paper9.pdf (дата обращения: 15.12.2018).

[17] *Мельников Б.Ф., Тренина М.А.* Об одной задаче восстановления матриц расстояний между цепочками ДНК // International Journal of Open Information Technologies. 2018. Т. 6, № 6. С. 1-13. URL: https://elibrary.ru/item.asp?id=35050442 (дата обращения: 15.12.2018).

[18] *Melnikov B.F., Trenina M.A., Kochergin A.S.* On one problem of reconstructing matrix distances between chains of DNA // IFAC-PapersOnLine. 2018. Vol. 51, Issue 32. Pp. 378-383. DOI: 10.1016/j.ifacol.2018.11.413

[19] *Мельников Б.Ф., Тренина М.А.* Применение метода ветвей и границ в задаче восстановления матрицы расстояний между цепочками ДНК // International Journal of Open Information Technologies. 2018. Т. 6, № 8. С. 1-13. URL: https://elibrary.ru/item.asp?id=35392804 (дата обращения: 15.12.2018).

[20] *Melnikov B.F., Trenina M.A.* On possible methods for solving the problem of reconstructing the matrix of distances between DNA strings // CEUR Workshop Proceedings. 2018. Vol. 2258. Pp. 11-20. URL: http://ceur-ws.org/Vol-2258/paper2.pdf (дата обращения: 15.12.2018).

[21] *Гудман С., Хидетниеми С.* Введение в разработку и анализ алгоритмов. М.: Мир, 1981. 364 с.

[22] *Громкович Ю.* Теоретическая информатика. Введение в теорию автоматов, теорию вычислимости, теорию сложности, теорию алгоритмов, рандомизацию, теорию связи и криптографию. СПб: Невский Диалект, БХВ-Петербург, 2010. 338 с.

[23] *Needleman S., Wunsch Ch.* A general method applicable to the search for similarities in the amino acid sequence of two proteins // Journal of Molecular Biology. 1970. Vol. 48, no. 3. Pp. 443-453. DOI: 10.1016/0022-2836(70)90057-4

[24] *Мельников Б.Ф., Тренина М.А., Кочергин А.С.* Подход к улучшению алгоритмов расчёта расстояний между цепочками ДНК (на примере алгоритма Нидлмана-Вунша) // Известия высших учебных заведений. Поволжский регион. Физико-математические науки. 2018. № 1(45). С. 46-59. DOI: 10.21685/2072-3040-2018-1-4.

[25] *Ayala F.J., Kiger J.A. Jr.* Modern Genetics (Second edition). Menlo Park and London, 1984. 923 pp.

Поступила 15.12.2018; принята к публикации 20.01.2019;
опубликована онлайн 19.04.2019.

Об авторах:
Абрамян Михаил Эдуардович, доцент, Институт математики, механики и компьютерных наук им. И.И. Воровича, Южный федеральный университет (344006, Россия, г. Ростов-на-Дону, ул. Б. Садовая, д. 105/42), кандидат физико-математических наук, ORCID: http://orcid.org/0000-0002-2802-6144, m-abramyan@yandex.ru
Мельников Борис Феликсович, профессор, кафедра информационных систем, сетей и безопасности, факультет информационных технологий, Российский государственный социальный университет (129226, Россия, г. Москва, ул. Вильгельма Пика, д. 4, стр. 1), доктор физико-математических наук, ORCID: http://orcid.org/0000-0002-6765-6800, bf-melnikov@yandex.ru
Тренина Марина Анатольевна, старший преподаватель, кафедра прикладной математики и информатики, Тольяттинский государственный университет (445020, Россия, г. Тольятти, ул. Белорусская, д. 14), ORCID: http://orcid.org/0000-0002-0580-5911, trenina.m.a@yandex.ru

Все авторы прочитали и одобрили окончательный вариант рукописи.

References

[1] Toppi J., Vico Fallani F.De, Petti M., Vecchiato G., Maglione A.G., Cincotti F., Salinari S., Mattia D., Babiloni F., Astolfi L. A new statistical approach for the extraction of adjacency matrix from effective connectivity networks. Proceedings of the 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), Osaka, 2013, pp. 2932-2935. (In Eng.) DOI: 10.1109/EMBC.2013.6610154

[2] Zhou J., Zhong P., Zhang T. A Novel Method for Alignment-free DNA Sequence Similarity Analysis Based on the Characterization of Complex Networks. *Evolutionary Bioinformatics*. 2016; 12:229-235. (In Eng.) DOI: 10.4137/EBO.S40474

[3] Alexeev N., Aidagulov R., Alekseyev M.A. A Computational Method for the Rate Estimation of Evolutionary Transpositions. In: Ortuño F., Rojas I. (eds) Bioinformatics and Biomedical Engineering. IWBBIO 2015. Lecture Notes in Computer Science. Springer, Cham. 2015; 9043:471-480. (In Eng.) DOI: 10.1007/978-3-319-16483-0_46

[4] Evdokimova N.I., Kuznetsov A.V. Local patterns in the copy-move detection problem solution. *Computer Optics*. 2017; 41(1):79-87. (In Russ.) DOI: 10.18287/2412-6179-2017-41-1-79-87

[5] Eckes B., Nischt R., Krieg T. Cell-matrix interactions in dermal repair and scarring. *Fibrogenesis and Tissue Repair*. 2010; 3(4). 11 pp. (In Eng.) DOI: 10.1186/1755-1536-3-4

[6] Korabelshchikova S.Y., Melnikov B.F., Pivneva S.V., Zyablitseva L.V. Linear codes and some their applications. *Journal of Physics: Conference Series*. 2018; 1096:012174. (In Eng.) DOI: 10.1088/1742-6596/1096/1/012174

[7] Korabelshchikova S.Y., Melnikov B.F., Pivneva S.V., Zyablitseva L.V. Linear error correcting codes and their application in DNA analysis. Proceedings of the 4th International Conference on Information Technology and Nanotechnology (ITNT-2018). Samara, 2018, pp. 2095-2100. Available at: https://elibrary.ru/item.asp?id=34894963& (accessed 15.12.2018). (In Eng.)

[8] Melnikov B., Radionov A., Gumayunov V. Some Special Heuristics for Discrete Optimization Problems. Proceedings of the Eighth International Conference on Enterprise Information Systems: Databases and Information Systems Integration (ICEIS- 2006), Paphos, Cyprus, May 23-27, 2006, pp. 360-364. (In Eng.)

[9] Melnikov B. Discrete optimization problems - some new heuristic approaches. Proceedings of the Eighth International Conference on High-Performance Computing in Asia-Pacific Region (HPCASIA'05), Beijing, 2005, pp. 8 pp.-82. (In Eng.) DOI: 10.1109/HPCASIA.2005.34

[10] Melnikov B.F., Melnikova E.A., Pivneva S.V., Dudnikov V.A., Davydova E.V. Geometric and game approaches for some discrete optimization problems. *CEUR Workshop Proceedings*. 2018; 2212:312-321. Available at: http://ceur-ws.org/Vol-2212/paper42.pdf (accessed 15.12.2018). (In Eng.)

[11] Forrest A.R.R. et al. A promoter-level mammalian expression atlas. *Nature*. 2014; 507:462-470. (In Eng.) DOI: 10.1038/nature13182

[12] Klyosov A.A., Faleeva T. Excavated DNA from Two Khazar Burials. *Advances in Anthropology*. 2017; 7:17-21. (In Eng.) DOI: 10.4236/aa.2017.71002

[13] Melnikov B.F., Pevneva S.V., Trifonov M.A. Multiheuristic approach to compare the quality of defined metrics on the set of dna sequences. *Sovremennye informacionnye tehnologii i IT-obrazovanie* = Modern Information Technologies and IT-Education. 2017; 13(2):89-96. (In Russ.) DOI: 10.25559/SITITO.2017.2.235

[14] Melnikov B.F., Pivneva S.V., Trifonov M.A. Comparative analysis of the algorithms of calculating distances of DNA sequences and some related problems. Proceedings of the Third International Conference on Information Technology and Nanotechnology (ITNT-2017). Samara, 2017, pp. 1640-1645. Available at: https://elibrary.ru/item.asp?id=29267020 (accessed 15.12.2018). (In Eng.)

[15] Melnikov B.F., Pivneva S.V., Trifonov M.A. Various algorithms, calculating distances of DNA sequences, and some computational recommendations for use such algorithms. *CEUR Workshop Proceedings*. 2017; 1902:43-47. Available at: http://ceur-ws.org/Vol-1902/paper8.pdf (accessed 15.12.2018). (In Eng.)

[16] Melnikov B.F., Melnikova E.A., Pivneva S.V., Trenina M.A. An approach to analysis of the similarity of DNA-sequences. *CEUR Workshop Proceedings*. 2018; 2212:63-72. Available at: http://ceur-ws.org/Vol-2212/paper9.pdf (accessed 15.12.2018). (In Eng.)

[17] Melnikov B.F., Trenina M.A. On a problem of the reconstruction of distance matrices between DNA sequences. *International Journal of Open Information Technologies*. 2018; 6(6):1-13. Available at: https://elibrary.ru/item.asp?id=35050442 (accessed 15.12.2018). (In Russ.)

[18] Melnikov B.F., Trenina M.A., Kochergin A.S. On one problem of reconstructing matrix distances between chains of DNA. *IFAC-PapersOnLine*. 2018; 51(32):378-383. (In Eng.) DOI: 10.1016/j.ifacol.2018.11.413

[19] Melnikov B.F., Trenina M.A. The application of the branch and bound method in the problem of reconstructing the matrix of distances between DNA strings. *International Journal of Open Information Technologies*. 2018; 6(8):1-13.

Available at: https://elibrary.ru/item.asp?id=35392804 (accessed 15.12.2018). (In Russ.)

[20] Melnikov B.F., Trenina M.A. On possible methods for solving the problem of reconstructing the matrix of distances between DNA strings. *CEUR Workshop Proceedings*. 2018; 2258:11-20. Available at: http://ceur-ws.org/Vol-2258/paper2.pdf (accessed 15.12.2018). (In Eng.)

[21] Goodman S.E., Hedetniemi S.T. Introduction to the Design and Analysis of Algorithms. NY: McGraw-Hill, 1977. 371 pp. (In Eng.)

[22] Hromkovič J. Algorithmics for Hard Problems. Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics. Springer-Verlag Berlin Heidelberg, 2004. 538 pp. (In Eng.) DOI: 10.1007/978-3-662-05269-3

[23] Needleman S., Wunsch Ch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*. 1970; 48(3):443-453. (In Eng.) DOI: 10.1016/0022-2836(70)90057-4

[24] Melnikov B.F., Trenina M.A., Kochergin A.S. An approach to improving algorithms for computing distances between DNA chains (by the example of the Needleman – Wunsch algorithm). *University proceedings. Volga region. Physical and mathematical sciences*. 2018; 1(45):46-59. (In Russ.) DOI: 10.21685/2072-3040-2018-1-4.

[25] Ayala F.J., Kiger J.A. Jr. Modern Genetics (Second edition). Menlo Park and London, 1984. 923 pp. (In Eng.)

Submitted 15.12.2018; revised 20.01.2019;
published online 19.04.2019.

About the authors:

Mikhail E. Abramyan, Associate Professor, Institute of Mathematics, Mechanics, and Computer Science named after of I.I. Vorovich, Southern Federal University (105/42 Bolshaya Sadovaya St., Rostov-on-Don 344006, Russia), Ph.D. (Phys.-Math.), ORCID: http://orcid.org/0000-0002-2802-6144, m-abramyan@yandex.ru

Boris F. Melnikov, Professor, Department of Information Systems, Networks and Safety, Faculty of Information Technology, Russian State Social University (4 Wilhelm Pieck St., build 1, Moscow 129226, Russia), Dr. Sci. (Phys.-Math.), ORCID: http://orcid.org/0000-0002-6765-6800, bf-melnikov@yandex.ru

Marina A. Trenina, senior lecture, Department of Applied Mathematics and Computer Science, Togliatti State University (14 Belorusskaya St., Togliatti 445020, Russia), ORCID: http://orcid.org/0000-0002-0580-5911, trenina.m.a@yandex.ru

All authors have read and approved the final manuscript.